Remarks

Applicants respectfully request reconsideration of the present application in view of the foregoing amendments and the following remarks. Claims 7, 14, 15, 21, 24, 27-34, 41-46, and 53-60 are pending in the application. Claims 7, 14, 15, 21, 24, 27-34, 41-46, and 53-60 are rejected. No claims have been allowed. Claims 7, 24, 30, 34, and 53 are independent. Claims 1-6, 8-13, 16-20, 22, 23, 25, 26, 33, 35-40, and 47-52 have been canceled without prejudice. Claim 61 is new.

Cited Art

The Action cites Evans et al., "Splint Manual, Version 3.1.1-1," June 5, 2003 ("Splint").

Splint Manual Disputed as Prior Art

The Action rejects claims 7, 14, 15, 21, 24, 27-34, 41-46, and 53-60 under 35 U.S.C. § 102(a) as allegedly being anticipated by the Splint Manual. The Splint Manual is dated "5 June 2003." Applicants do not admit that this Splint manual is prior art to the present application and reserve the right to provide evidence of prior conception.

Claim Rejections under 35 USC § 102

The Action rejects claims 7, 14, 15, 21, 24-37, and 41-55 under 35 U.S.C. § 102(a) as allegedly being anticipated by the Splint Manual.

For a rejection under 35 U.S.C. § 102 to be proper, the applied art must show each and every element as set forth in a claim. (*See* MPEP § 2131.) Applicants respectfully submit that the claims in their present form are allowable over the applied art because it does not teach or suggest all the claim limitations of the claims.

Applicants respectfully traverse these rejections and submit that the claims in their present form are allowable over the applied art.

Independent Claim 7

Independent claim 7 recites in part: "*wherein the annotations on the first pointer are placed in an argument list to a function call that uses the first pointer as a parameter.*"

The applied art does not teach or suggest, e.g., the above-cited language of independent claim 7.

In the response to arguments, the Action states: "Regarding claim 7, the examiner asserts that the Splint manual discloses, "'wherein the annotations on the first pointer are placed in an argument list to a function call that uses the first pointer as a parameter.' In the examples on p. 49, the annotations occur between the function name and the semicolon indicating the end of the function call, and thus may be considered part of the argument list."

Applicants respectfully disagree. The Examiner is stating that anything at all within a function is part of an argument list to the function (e.g., "annotations occur between the function name and the semicolon indicating the end of the function call"). The splint syntax, as defined in the Splint Manual clearly defines an argument list (called a parameter-type-list) separately from the rest of the function.

The Splint manual includes C syntax grammar "modified to show the syntax of syntactic comments." The definition of a function, from the Splint manual, page 103, is shown below."

Functions

direct-declarator ⇒
direct-declarator (*parameter-type-list*$_{opt}$) *stateClause**$_{opt}$ *globals*$_{opt}$ *modifies*$_{opt}$
| *direct-declarator* (*identifier-list*$_{opt}$) *stateClause**$_{opt}$ *globals*$_{opt}$ *modifies*$_{opt}$

stateClause ⇒ /*@ (uses | sets | defines | allocates | releases) *reference*,* ;$_{opt}$ @*/
| /*@ (ensures | requires) *stateTag reference*,* ;$_{opt}$ @*/ (Section 7.4)

stateTag ⇒ only | shared | owned | dependent | observer | exposed | isnull | notnull
| *identifier* (Annotation defined by metastate definition, Section 10)

This shows that a function call is followed by an argument list ("parameter-type-list"). This argument list does not contain any of the Splint annotations; e.g., a state clause. The state clause, rather, clearly is placed **after** the argument list. As the Splint manual very clearly shows that Splint annotations are never *"placed in an argument list to a function call,"* Splint does not teach or show the claim 1 language "wherein the annotations on the first pointer are placed in an argument list to a function call…." As

such, Splint even further does not teach or suggest the additional features of claim 1, "wherein the annotations on **the first pointer are placed in an argument list to a function call *that uses the first pointer as a parameter*.**" For at least this reason, claim 7 is in condition for allowance.

Moreover, Applicants respectfully request that the Examiner produce authority for his statement that "the annotations occur between the function name and the semicolon indicating the end of the function call, and thus may be considered part of the argument list." As stated earlier, Applicants respectfully disagree. The Examiner is stating that anything at all within a function call is part of an argument list to the function (e.g., "annotations occur between the function name and the semicolon indicating the end of the function call"). Applicant respectfully believes that this is simply not true. Specifically, Applicants respectfully request that the Examiner produce authority that the entire function to the end deliminator (the semicolon) is part of the argument list. "See Zurko, 258 F.3d at 1386, 59 USPQ2d at 1697 ("[T]he Board [or examiner] must point to some concrete evidence in the record in support of these findings" to satisfy the substantial evidence test). If the examiner is relying on personal knowledge to support the finding of what is known in the art, the examiner must provide an affidavit or declaration setting forth specific factual statements and explanation to support the finding. See 37 CFR 1.104(d)(2)." MPEP 2144.03.

Claim 7 is allowable. Claims 14, 15 and 21 depend from claim 7 and are allowable for at least the reasons given above in support of claim 7. Therefore, the rejections of claims 7, 14, 15 and 21 under 35 U.S.C. § 102 should be withdrawn. Such action is respectfully requested. Further, claims 14, 15, and 21 recite separately patentable combinations.

Independent Claim 24

Independent claim 24 recites in part:

> wherein the annotation comprises a first instance of a keyword, the first instance of the keyword indicating that the first value satisfies all usability properties necessary to allow a first function to rely on the first value, wherein other instances of the keyword identical to the first instance are operable to indicate that other values having different respective value types satisfy all usability properties necessary to allow functions to rely on the respective other values, wherein use of the keyword associates a pre-determined set of usability properties

with a value type, and wherein the usability properties depend on the value type. [Emphasis added.]

The Splint Manual does not teach or suggest the above-cited language of independent claim 24. In particular, the applied art does not teach or suggest "an annotation that comprises a first instance of a keyword that indicates the first value satisfies all usability properties necessary to allow a first function to rely on the first value, wherein other instances of the keyword identical to the first instance are operable to indicate that other values having different respective value types satisfy all usability properties necessary to allow functions to rely on the respective other values, wherein use of the keyword associates a pre-determined set of usability properties with a value type, and wherein the usability properties depend on the value type."

The Examiner rejects the above-quoted portion of the claim stating: "the Splint manual discloses, "wherein use of the keyword associates a pre-determined set of usability properties with a value type.: See, e.g., section 4, beginning on p. 19, describing annotations associated with types and the corresponding checking associated with various type-specific annotations." [Action, page 2.] This same assertion is repeated to reject the entire above-cited language of claim 24. [Action, p. 5] Applicants have read the cited materials and cannot locate, e.g., the claim 24 language "wherein the annotation comprises a first instance of a keyword, the first instance of the keyword indicating that the first value satisfies all usability properties necessary to allow a first function to rely on the first value, wherein other instances of the keyword identical to the first instance are operable to indicate that other values having different respective value types satisfy all usability properties necessary to allow functions to rely on the respective other values."

As a representative instance, applicants cannot locate "a first instance of a keyword, the first instance of the keyword indicating that the first value satisfies all usability properties necessary to allow a first function to rely on the first value…." The Splint manual discusses the opposite, that is, it discusses instances where values are suspect. Splint, in the asserted matter, reports where one is **not allowed** to rely on a value. For example, section 4 describes different types and the sorts of type checking that can be done on them using Splint annotations. This type checking is negative, in that it reports errors. The Applicants' representative has carefully read the section and finds no location where the opposite holds, that is, a keyword that indicates that the "first value satisfies all usability properties necessary to allow a first function to rely on

the first value…." For example, "Splint supports three different kinds of arbitrary integral types." [Splint, 20.] These three types allow different types of integers to be assigned to a single variable. "Splint reports an error if the code depends on the actual representation of a type declared as an arbitrary integral." [*Id.*] That is, rather than indicating that a "first value satisfies all usability properties…" Splint makes the **opposite** assumption, that an error might exist, and so checks to see if the value is allowable. If there is an error, Splint then "reports an error if the code depends on the actual representation of a type declared as an arbitrary integral." Splint also has abstract types, but, similar to arbitrary integral types, "Splint reports an error if an instance of [an abstract type is passed incorrectly.]" [Splint, pages 21-22.]

Splint's typechecking, which uniformly reports an error if a defined type is misused, teaches away from "a first instance of a keyword, the first instance of the keyword indicating that the first value satisfies all usability properties necessary to allow a first function to rely on the first value…." As such, Splint also does not discuss, and teaches away from the additional claim 24 language "wherein other instances of the keyword identical to the first instance are operable to indicate that other values having different respective value types satisfy all usability properties necessary to allow functions to rely on the respective other values…." For at least this reason, claim 24 is in condition for allowance. Claims 25-29 depend from claim 24 and are allowable for at least the reasons given above in support of claim 24. Therefore, the rejection of claims 24-29 under 35 U.S.C. § 102 should be withdrawn. Such action is respectfully requested. Claims 25-29 also recite separately patentable combinations.

Independent Claim 30

Amended independent claim 30 recites:

> 30. (Currently Amended) In a computer system, a method of annotating computer program code stored on a computer-readable medium, wherein the computer program code is operable to cause a computer to perform according to instructions in the computer program code, the method comprising:
>
> inserting an annotation having an argument comprising a second value type in the computer program code, wherein the annotation annotates a variable having a first value type;
>
> wherein the annotation changes the first value type of the variable to the second value type of the annotation argument.
>
> [Emphasis added.]

This is described in the Specification, at, e.g., page 20 line 13 through page 21, line 9 with reference to the "typefix" property. For example, at page 20, the Application states, "the typefix property can be used to override the declared C/C++ type. The interpretation of *valid* for *the annotated value* is obtained from the type given by the typefix instead of the declared C/C++ type." [Emphasis added.] Table 11 on page 21 of the Specification gives describes the Typefix property as follows: "Annotates any value. States that if the value is annotated as *valid*, then it satisfies all of the properties of valid *ctype* values. *ctype* must be a visible C/C++ type at the program point where the annotation is placed."

The Splint manual does not teach or suggest, e.g.,

> inserting an annotation having an argument comprising a second value type in the computer program code, wherein the annotation annotates a variable having a first value type;
>
> wherein the annotation changes the first value type of the variable to the second value type of the annotation argument. [Emphasis added.]

.Regarding unamended claim 30, the Examiner states, "As per claim 30, [Splint] discloses inserting an annotation . . . wherein the annotation indicates that the value has usability properties that depend on the properties of a second value type denoted by the argument of the annotation." [*See* Action at p. 6.] Specifically, the Examiner cites the "alt" notation. reading "see, e.g., pp. 24 and 57, describing the alt annotation, the alt annotation overrides the type checking by adding alternative types, which incorporates they type checking associated with the newly added types. [Action, pages 6-7.]

In contrast, the "alt" notation described in the Splint manual only indicates that a declaration may have types added to it, not that the original type can be changed. [See, e.g, pg. 24, reading "The /*@alt type,+@*/ annotation creates a union type. For example, **int** /*(**alt char**, **unsigned char**@*/ **c** declares **c** such that either an **int**, **char** or **unsigned char** value may be assigned to it without warning."] The original type here is **int**. *alt* does not change the ability of the variable **c** to hold an **int** type, rather it adds valid types—here, **char** and **unsigned char**—to the values that **c** can hold. This does not describe, but rather teaches against changing a variable defined as one type to another type.

As another example, the alt notation can be used to add the type *float* to a type already declared as *int*. [*See* Splint manual at page 57.] The original variable still can hold the value *int,* however. As such, Split teaches against the amended claim 30 language "inserting an annotation

having an argument comprising a second value type in the computer program code, wherein the annotation annotates a variable having a first value type; wherein the annotation changes the first value type of the variable to the second value type of the annotation argument"

Claim 30 is allowable. Claims 31-32 depend from claim 30 and are allowable for at least the reasons given above in support of claim 30. Therefore, the rejection of claims 30-32 under 35 U.S.C. § 102 should be withdrawn. Such action is respectfully requested. Additionally, claims 31 and 32 recite separately patentable combinations.

Independent Claim 34

Independent claim 34 recites in part:

> adding an annotation to a pointer in the computer program code, wherein the annotation describes transferring buffer properties from a second pointer to the pointer; and including a location parameter with the annotation, wherein the location parameter describes the logical buffer pointed to by the pointer. [Emphasis added.]

This is described in the Specification at, e.g., page 17, lines 10-14, reproduced below:

> The *aliased* property is useful for transferring buffer properties from one pointer to another. *notAliased* is useful for guaranteeing that two buffers are not aliased (i.e., that two buffers do not overlap). The *aliased* property is described in Table 8 below.

Property	Meaning
aliased(location)	Annotates a buffer pointer and states that the pointer points into the same logical buffer as *location*. The pointers need not be equal.

Table 8: The *aliased* property

The Splint manual does not teach or suggest the claim 34 language, above. To allegedly teach or suggest, the Action cites to section 6 of the Splint manual. [Action, page 3.] Section 6 describes "detecting errors involving dangerous aliasing of parameters." Sections 6.1 through 6.2.2 described the various types of errors that can be so detected. Nowhere in the section does a Splint annotation transfer any properties. They just **detect** when properties have been **transferred**. As such they teach away from actually performing any sort of transfer, let alone the specific transfers in claim 34 such as the language "transferring buffer properties from a second pointer to the pointer" and so even more emphatically fails to teach or suggest the

additional limitations "adding an annotation to a pointer in the computer program code, wherein the annotation describes transferring buffer properties from a second pointer to the pointer; and including a location parameter with the annotation, wherein the location parameter describes the logical buffer pointed to by the pointer."

For at least this reason claim 34 is allowable. Claims 35-37 depend from claim 34 and are allowable for at least the reasons given above in support of claim 34. Therefore, the rejection of claims 34-37 under 35 U.S.C. § 102 should be withdrawn. Such action is respectfully requested. Claims 35-37 also recited separately patentable combinations.

In the event that the Office maintains the rejection of independent claim 34 under 35 U.S.C. §102, Applicant respectfully requests that the Office, in the interests of compact prosecution, identify on the record and with specificity sufficient to support a prima facie case of anticipation, where in the Splint reference the subject feature of independent claim 34 of "transferring buffer properties" is alleged to be taught.

Independent Claim 53

Independent claim 53 recites in part:

> reading at least one annotation having an argument from the annotated computer program code, wherein the at least one annotation annotates a value having a first declared value type with a first set of usability properties used only in an annotation context, and wherein the annotation overrides the first set of usability properties used only in the annotation context of the first declared value type and indicates a second set of usability properties used only in the annotation context for the value that depend on the second value type denoted by the argument of the annotation, such that the second set of usability properties are used in the context of a second annotation rather than the first set of usability properties… [Emphasis added.]

This is supported in the Specification, at, e.g., page 18 line 3 through page 20, line 12, with reference to the "valid" property, and at page 19, table 10, discussing the interpretations of the "valid" property for different value types.

Splint does not teach or suggest the claim 53 language "reading at least one annotation having an argument from the annotated computer program code, wherein the at least one annotation annotates a value having a first declared value type with a first set of usability properties used only in an annotation context…." Regarding claim 53, the Examiner states, "see,

e.g., pp. 24 and 57, describing the alt annotation; the alt annotation overrides the type checking by adding alternative types, which incorporates the type checking associated with the newly added types." [*See* Action at p. 8.]

The "alt" notation described in the Splint manual only indicates that a declaration of one type may have other types added to it. [See, e.g, pg. 24, reading "The /*@alt type,+@*/ annotation creates a union type. For example, **int** /*(**alt char**, **unsigned char**@*/ **c** declares **c** such that either an **int**, **char** or **unsigned char** value may be assigned to it without warning.] This does not teach or suggest that the original type of the variable can be changed. The types that may be added to a variable using the "alt" notation in Splint are the standard types available in the language. This is reinforced by the syntax of the alt type shown on page 103. It declares an alt type as "/*@alt basic-type, +@*/" As an alt type can only be a **basic type** of the language, and as the alt type only adds types to the type a variable was previously declared, it does not provide annotations "**used only in an annotation context**." As such, Split strongly teaches against the claim 30 language "reading at least one annotation having an argument from the annotated computer program code, wherein the at least one annotation annotates a value having a first declared value type with a first set of usability properties used only in an annotation context…."

For at least this reason, claim 53 is allowable. Claims 54-55 depend from claim 53 and are allowable for at least the reasons given above in support of claim 53. Therefore, the rejection of claims 54-55 under 35 U.S.C. § 102 should be withdrawn. Such action is respectfully requested. Moreover, claims 54 and 55 recite separately patentable combinations.

In the event that the Office maintains the rejection of independent claim 53 under 35 U.S.C. §102, Applicant respectfully requests that the Office, in the interests of compact prosecution, identify on the record and with specificity sufficient to support a prima facie case of anticipation, where in the Splint reference the subject feature of independent claim 53 of "a first set of usability properties used only in an annotation context" is alleged to be taught.

Support for the Amendments

Support for the Amendments can be found in the specification and figures as originally filed. Further, the following specific examples are given:

Specification, page 20, line 19 to page 21, line 9.

Table 9, Table 10.

Interview Request

If the claims are not found by the Examiner to be allowable, the Examiner is requested to call the undersigned attorney to set up an interview to discuss this application.

Conclusion

The claims in their present form should be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 595-5300
Facsimile: (503) 595-5301

By /Genie Lyons/
Genie Lyons
Registration No. 43,841